Lab Exercise 3: Reading Files
CS 2334

September 8, 2016

Introduction

There are many reasons to bring data from a file into a program. Once the data from the file are represented within a data structure, they can be analyzed or presented to a user, and can even be used to take other external actions. Importing data from a file is, therefore, a useful skill that you will employ for the rest of your academic and professional career as a computer scientist.

In this laboratory, we will load data from a file into a data structure, analyze it, and then display the information in a structured form to the user. We have provided a specification for three classes, as well as a simple test that you can check your output. Your task is to complete the implementation of the classes according to the specification that we provide.

Learning Objectives

By the end of this laboratory exercise, you should be able to:

1. read and understand method-level specifications (including from UML diagrams),
2. read data from the file,
3. complete the implementation of a class containing multiple instance variables and methods,
4. use the information in the file to create an array of objects, and

5. scan through an array of objects and display the items that match a given criterion.

Proper Academic Conduct

This lab is to be done individually. Do not look at or discuss solutions with anyone other than the instructor or the TAs. Do not copy or look at specific solutions from the net.

Preparation

1. Download:
`http://www.cs.ou.edu/~fagg/classes/cs2334/labs/lab3/lab3.zip`
2. *File* menu: *Import*
3. Select *General/Existing Projects into Workspace* and then click *Next*
4. *Select archive file*: browse to the lab3.zip file
5. Click *Finish*
6. Once you create the new project, it will not initially know where to find the standard Java libraries (it varies depending on your configuration). In one of the provided Java files, find an undefined reference to a standard class (e.g., String) and mouse over it. Java will provide a list of ways to repair the problem:
   (a) Select *Fix project setup*
   (b) Select *Add library: JRE System Library*
   (c) Click *OK*

Lab 3

For this lab, you will be parsing a file that contains a list of Pokemon compiled by the Kaggle website.[1] This file contains various statistics about each of the Pokemon. The file encodes this information in a table using the *comma separated values* (CSV) format. If you double click on this file from within Eclipse, it will attempt to open the file in a spreadsheet program, such as Excel. Alternatively, you can select the file and *open with* a text editor. This will open the raw file in Eclipse.

Each line of this file encodes information about exactly one Pokemon. Each Pokemon is described using thirteen distinct values that are separated by commas in the file. These are:

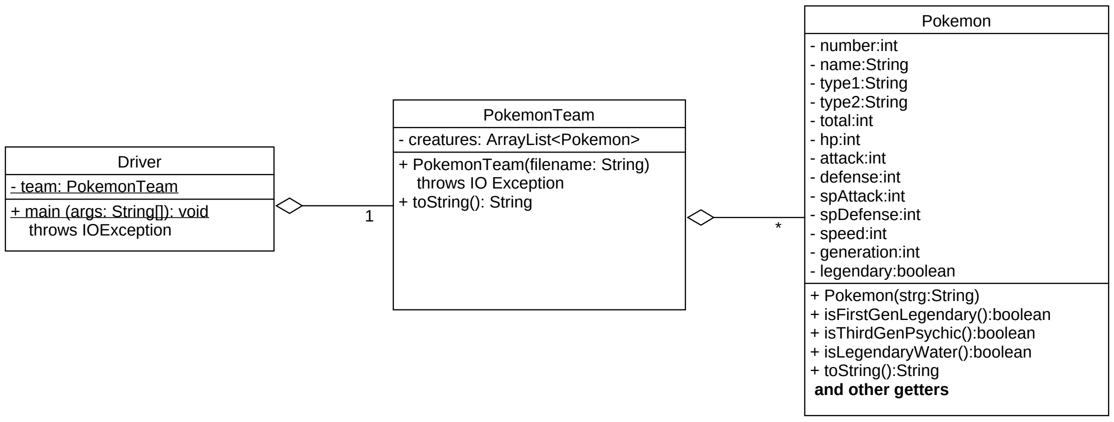
1. The number of the Pokemon. It is important to notice that the number is not unique to each Pokemon. For example, the three different evolutions of Charizard all share the number *6*. This will not affect the outcome of this project.
2. The name of the Pokemon. Again, this is not unique.
3. The first type of the Pokemon.
4. The second type of the Pokemon. This may be blank.
5. The total of all the following statistics.
6. The health points (HP) of the Pokemon.
7. The base modifier of a normal attack.
8. The base damage resistance against normal attacks.
9. The base modifier for special attacks.
10. The base damage resistance against special attacks.
11. The speed of the Pokemon.
12. The generation the Pokemon was introduced.
13. Whether or not a Pokemon is legendary.

Your task for this lab is to load these Pokemon from a specified file, store this list of Pokemon and then scan the list for Pokemon that match certain criteria.

[1] `https://www.kaggle.com/abcsds/pokemon`

Classes

The following UML diagram summarizes the classes that are to be implemented for this laboratory exercise.



The *Pokemon* class is responsible for representing a single Pokemon. Here are the methods that need to be implemented:

- *Pokemon* constructor. Here you will need to take in a String that corresponds to one Pokemon and parse it to populate the class variables.
- *isFirstGenLegendary()* This method indicates whether a *Pokemon* object is in the first generation and is legendary.
- *isThirdGenPsychic()* This method indicates whether the Pokemon is in the third generation and is of the pyschic type.
- *isLegendaryWater()* This method indicates whether the Pokemon is legendary and is of the water type.
- *toString()*. This method returns a String in the format shown in *lab3-expectedOutput.pdf*.
- A full set of getters. Use Eclipse to generate these for you.

The *PokemonTeam* class represents a team of *Pokemon*. Here are the methods that need to be implemented:

- *PokemonTeam* constructor. This method takes as input a file name, reads through the file, creates one *Pokemon* object from each line in the file, and adds each to an ArrayList.

- *toString()*. This method returns a String in the format shown in *lab3-expectedOutput.pdf*.

The *Driver* class contains a PokemonTeam object and the main() method. The main() method of this class:

- Creates a *PokemonTeam* object with a specific file name
- Prints out the *PokemonTeam* object

Reading a File

Within the PokemonTeam constructor, a file must be read and then parsed. An example of the code needed to pull out individual lines and add them to a list is provided below:

```
1 ArrayList<String> list = new ArrayList<String>();  // ArrayList of Strings
2 BufferedReader br = new BufferedReader (new FileReader("filename.txt"));
3
4 String strg = br.readLine();    // Read first line
5 while (strg != null) // Iterate as long as there is a next line
6 {
7     list.add(strg);     // Add the line to the ArrayList
8     strg = br.readLine();   // Attempt to get the next
9 }
```

(of course, you must create Pokemon object instances to add to your ArrayList).

A *BufferedReader* can take different input streams as a parameter. In Lab 1, an *InputStreamReader* was used in order to take input from *System.in* (the keyboard). In this lab, a *FileReader* will be used. A *FileReader* takes a file name as a parameter. Notice that the file name is a String[2] Line 2 of this example opens the file and turns it into a *FileReader* object that can then be used by the *BufferedReader* object.

Automatically Generating Getters and Setters in Eclipse

Eclipse is able to generate the getters and setters for a class for you. While editing the class, the steps are:

[2]Remember that String literals must have quotes around them.

- Declare all the instance variables
- Select *Source* on the toolbar
- Select *Generate Getters and Setters...* from the dropdown menu

From this menu, you can select the getters and setters you would like generated from the dropdown associated with each class variable. You can also select all getters, all setters or both. For this lab, remember that you are creating immutable classes.

Final Steps

1. Generate Javadoc using Eclipse.
    - Select *Project/Generate Javadoc...*
    - Make sure that your project is selected, as are the Driver, Pokemon, PokemonTeam classes
    - Select *Private* visibility
    - Use the default destination folder
    - Click *Finish*
2. Open the *lab3/doc/index.html* file using your favorite web browser or Eclipse (double clicking in the package explorer will open the web page). Check to make sure that that both of your classes are listed and that all of your documented methods have the necessary documentation.
3. If you complete the above instructions during lab, you may have your implementation checked by one of the TAs.

Submission Instructions

- All required components (source code and compiled documentation) are due at 11:59pm on Friday, September 9th.
- Method 1: Submit through Eclipse
    1. From the *Window* menu, select *Preferences/Configured Assignment.*
    2. Select your project.

3. From the Project menu, select *Submit Assignment*.
4. Under *Select the assignment to submit*, select *CS 2334/Lab 03 (2334): Pokemon*.
5. Click *Change Username or Password...*. Enter your Web-Cat username and password. Click *OK*. You should only need to do this step once per session.
6. Click *Finish*.
7. Your browser should automatically open a Web-Cat page that shows your submission being graded. After a short wait, the page will show a report of your submission. See the main class web page for a link that describes the Web-Cat output.

- Method 2: Submit directly to the Web-Cat server

1. From the File menu, select *Export*.
2. Select *Java/JAR File*. Click *Next*.
3. Select and expand your project folder.
4. Select your *src* and *doc* folders. **Do not include csv files.**
5. Select *Export Java source files and resources*.
6. Select an export destination location (e.g., your *Documents* folder/directory). This file should end in *.jar*
7. Select *Add directory entries*.
8. Click *Finish*.
9. In your web browser, login to the Web-Cat server.
10. Click the *Submit* button.
11. Browse to your jar file.
12. Click the *Upload Submission* button.
13. The next page will give you a list of all files that you are uploading. If you selected the correct jar file, then click the *Confirm* button.
14. Your browser will then open a Web-Cat page that shows your submission being graded. After a short wait, the page will show a report of your submission. See the main class web page for a link that describes the Web-Cat output.

References

- The API of the *BufferedReader* class can be found at:
  `http://docs.oracle.com/javase/8/docs/api/java/io/BufferedReader.html`

Rubric

The project will be graded out of 100 points. The distribution is as follows:

Correctness/Testing: 45 points

The Web-Cat server will grade this automatically upon submission. Your code will be compiled against a set of tests (called *Unit Tests*). These unit tests will not be visible to you, but the Web-Cat server will inform you as to which tests your code passed/failed. This grade component is proportional to the fraction of tests that your code passes (so 22.5 points means that your code passed half of the tests)

Style/Coding: 20 points

The Web-Cat server will grade this automatically upon submission. Every violation of the *Program Formatting* standard described in Lab 1 will result in a subtraction of a small number of points (usually two points). Looking at your submission report on the Web-Cat server, you will be able to see a notation for each violation that describes the nature of the problem and the number of subtracted points.

Design/Readability: 35 points

This element will be assessed by a grader (typically sometime after the lab deadline). Any *errors* in your program will be noted in the code stored on the Web-Cat server, and two points will be deducted for each. Possible errors include:

- Non-descriptive or inappropriate project- or method-level documentation
- Missing or inappropriate inline documentation
- Inappropriate choice of variable or method names
- Inefficient implementation of an algorithm
- Incorrect implementation of an algorithm
- Incomplete coverage of your Unit Tests. We expect that your unit tests will test all lines of your code

If you do not submit compiled Javadoc for your lab, 5 points will be deducted from this part of your score.

Note that the grader may also give *warnings* or other feedback. Although no points will be deducted, the issues should be addressed in future submissions(where points may be deducted).

Bonus: up to 5 points

You will earn one bonus point for every two hours that your assignment is submitted early.

Penalties: up to 100 points

You will lose ten points for every minute that your assignment is submitted late.